

3/5/1 (Item 1 from file: 351)
DIALOG(R) File 351:Derwent WPI
(c) 2000 Derwent Info Ltd. All rts. reserv.

010898234 **Image available**
WPI Acc No: 1996-395185/199640
XRPX Acc No: N96-333032

**Debugger system for debugging distributed target application system - has
debugger GUI and one or more debugger engines which may reside on local
or remote host computer**

Patent Assignee: SUN MICROSYSTEMS INC (SUNM)
Inventor: DAVIDSON A E; MASAMITSU J A
Number of Countries: 008 Number of Patents: 005
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week
EP 730227	A1	19960904	EP 96301430	A	19960301	199640 B
CA 2170724	A	19960904	CA 2170724	A	19960229	199701
JP 9120366	A	19970506	JP 9673240	A	19960304	199728
US 5819093	A	19981006	US 95399120	A	19950303	199847
US 6042614	A	20000328	US 95399120	A	19950303	200023
			US 985287	A	19980109	

Priority Applications (No Type Date): US 95399120 A 19950303; US 985287 A
19980109

Cited Patents: US 4589068; US 5371746

Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
EP 730227	A1	E	28	G06F-011/00	
Designated States (Regional): DE FR GB IT SE					
US 6042614	A			G06F-009/445	Cont of application US 95399120
JP 9120366	A		24	G06F-011/28	
CA 2170724	A			G06F-009/44	
US 5819093	A			G06F-009/455	

Abstract (Basic): EP 730227 A

The distributed debugger system includes a debugger GUI (502) which provides an interface with one or more debugger engines (dbx engines) for communicating with the dbx engines which reside on local and remote host computers. A communications mechanism is used by the dbx engines and the debugger GUI for sending and receiving messages from each other.

A remote dbx engine resides on a host computer and is connected to the debugger GUI by the communications mechanism. request and replies are made through an object request broker (ORB) that is aware of the locations and status of objects in the distributed object environment.

USE - Debugging new applications using objects in widely distributed, object-oriented, client server system.

ADVANTAGE - Enables developer to use objects and object implementations located on different host machine which is unknown to developer.

Dwg.14/15

Title Terms: SYSTEM; DEBUG; DISTRIBUTE; TARGET; APPLY; SYSTEM; ONE; MORE;
ENGINE; LOCAL; REMOTE; HOST; COMPUTER

Derwent Class: T01

International Patent Class (Main): G06F-009/44; G06F-009/445; G06F-009/455;
G06F-011/00; G06F-011/28

International Patent Class (Additional): G06F-009/06; G06F-012/00;
G06F-015/16

File Segment: EPI

3/5/2 (Item 1 from file: 347)
DIALOG(R) File 347:JAPIO
(c) 2000 JPO & JAPIO. All rts. reserv.

05505566

SYSTEM AND METHOD FOR DISTRIBUTED DEBUGGING OF DEBUGGING OF DISTRIBUTED
APPLICATION PROGRAM

PUB. NO.: 09-120366 JP 9120366 A]
PUBLISHED: May 06, 1997 (19970506)
INVENTOR(s): ANDORIYUU II DEBITSUDOSON
JIYON EI MASAMITSU
APPLICANT(s): SUN MICROSYST INC [198211] (A Non-Japanese Company or
Corporation), US (United States of America)
APPL. NO.: 08-073240 [JP 9673240]
FILED: March 04, 1996 (19960304)
PRIORITY: 7-399,120 [US 399120-1995], US (United States of America),
March 03, 1995 (19950303)
INTL CLASS: [6] G06F-011/28; G06F-009/06; G06F-009/44; G06F-012/00;
G06F-015/16
JAPIO CLASS: 45.1 (INFORMATION PROCESSING -- Arithmetic Sequence Units);
45.2 (INFORMATION PROCESSING -- Memory Units); 45.4
(INFORMATION PROCESSING -- Computer Applications)

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開平9-120366

(43) 公開日 平成9年(1997)5月6日

(51) Int.Cl. ⁶	識別記号	庁内整理番号	F I	技術表示箇所
G 0 6 F 11/28		7313-5B	G 0 6 F 11/28	A
9/06	5 3 0		9/06	5 3 0 Q
9/44	5 3 5		9/44	5 3 5
12/00	5 4 5		12/00	5 4 5 A
15/16	3 7 0		15/16	3 7 0 N

審査請求 未請求 請求項の数25 F D (全 24 頁)

(21) 出願番号 特願平8-73240

(22) 出願日 平成8年(1996)3月4日

(31) 優先権主張番号 08/399120

(32) 優先日 1995年3月3日

(33) 優先権主張国 米国 (US)

(71) 出願人 591064003

サン・マイクロシステムズ・インコーポレーテッド

SUN MICROSYSTEMS, INCORPORATED

アメリカ合衆国 94043 カリフォルニア州・マウンテンビュー・ガルシア アヴェニュー・2550

(72) 発明者 アンドリュー・イー・デビッドソン

アメリカ合衆国 95006 カリフォルニア州・ブルダー クリーク・ホプキンス ガルチ・1222

(74) 代理人 弁理士 山川 政樹

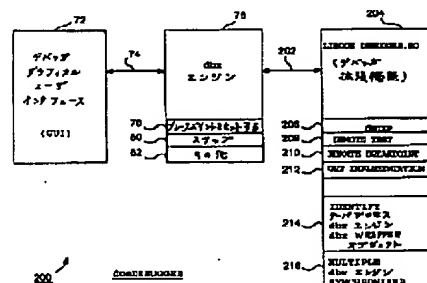
最終頁に続く

(54) 【発明の名称】 分散アプリケーション・プログラムをデバッグする分散デバッグのためのシステムおよび方法

(57) 【要約】 (修正有)

【課題】 分散アプリケーションのデバッグを容易に行えるようにする。

【解決手段】 アプリケーションのプログラマ/開発者が1台のホストマシンのところにおいて、開発されるアプリケーションがプログラマ/開発者には未知の他のホストマシンにおかれていることのあるオブジェクトおよびオブジェクト・インプリメンテーションを利用する分散ターゲット・コンピュータ・アプリケーションをデバッグする分散デバッガ・システムにおいて、分散オブジェクト環境における要求及び応答は、オブジェクトの場所及び状況を認識しているオブジェクト・リクエスト・ブローカ (ORB) によって行われ、ORBを実施するのに適している1つのアーキテクチャが、共通オブジェクト・リクエスト・ブローカ・アーキテクチャ仕様によって与えられる。



【特許請求の範囲】

【請求項1】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、

デバッガGUIおよび1つまたは複数のデバッガ・エンジン（以下、「dbxエンジン」と呼ぶ）であって、前記デバッガGUIが前記dbxエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記dbxエンジンが前記ローカルおよび遠隔のコンピュータに常駐できるデバッガGUIおよび1つまたは複数のdbxエンジンと、メッセージを互いに送受信するために前記dbxエンジンと前記デバッガGUIが使用する通信機構と、前記の1つまたは複数のdbxエンジンの1つであり、前記ローカル・ホスト・コンピュータから離隔したホスト・コンピュータにあり、前記通信機構によって前記デバッガGUIに接続されている遠隔dbxエンジンとを備えている分散デバッガ・システム。

【請求項2】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、

デバッガGUIおよび1つまたは複数のデバッガ・エンジン（以下、「dbxエンジン」と呼ぶ）であって、前記デバッガGUIが前記dbxエンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記dbxエンジンが前記ローカルおよび遠隔のコンピュータに常駐できるデバッガGUIおよび1つまたは複数のdbxエンジンと、メッセージを互いに送受信するために前記dbxエンジンと前記デバッガGUIが使用する通信機構と、遠隔ホスト・コンピュータにある前記ターゲット・アプリケーション・システムの一部をデバッグする際に使用される新しいdbxエンジンを前記遠隔ホスト・コンピュータに作成するために前記デバッガGUIが使用するdbxWrapperFactory機構とを備えている分散デバッガ・システム。

【請求項3】 前記dbxエンジンの1つが前記dbxエンジンの他のものと、これらのdbxエンジンが異なるホスト・コンピュータにあるかどうかにかかわらず、通信することを可能とする第1の通信機構をさらに含んでいることを特徴とする、請求項2に記載の分散デバッガ・システム。

【請求項4】 前記デバッガGUIが前記dbxエンジンの1つに、前記dbxエンジンがどのホスト・コンピュータにあるかにかかわらず、フォーカスすることを可能とする第2の通信機構をさらに含んでいることを特徴とする、請求項2に記載の分散デバッガ・システム。

【請求項5】 前記ユーザが、前記dbxエンジンがど

のホスト・コンピュータにあるかにかかわらず、すべての活動dbxエンジンのリストを前記デバッガGUIから取得できるようにする第3の通信機構をさらに含んでいることを特徴とする、請求項2に記載の分散デバッガ・システム。

【請求項6】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジンにおいて、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「IDL」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）dstep機構を備えているdbxエンジン。

【請求項7】 前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「IDL」）生成コード機構のどれを無視すべきかを決定するremote surrogate code test機構をさらに含んでいることを特徴とする、請求項6に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジン。

【請求項8】 指定されたオブジェクトのホストIDおよびプロセスID（PID）を探し出すGetImplementation機構をさらに含んでいることを特徴とする、請求項6に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジン。

【請求項9】 遠隔dbxエンジンが作動しているかどうかを判定し、作動していない場合には、dbxWrapperFactoryオブジェクトの機能を使用してdbxエンジンを作成し、遠隔ターゲット機能に接続するIdentifyRemoteFunction機構をさらに含んでいることを特徴とする、請求項6に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジン。

【請求項10】 dbxエンジンが互いに通信を行えるようにする多重dbxエンジン同期機構をさらに含んでいることを特徴とする、請求項6に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジン。

【請求項11】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用するdbxエンジンにおいて、

前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・ア

3

アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「IDL」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）`dstep`機構と、

前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「IDL」）生成コード機構のどれを無視すべきかを決定する`remote surrogate code test`機構と、

ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構と、

指定されたオブジェクトのホストIDおよびプロセスID（PID）を探し出す`GetImplementation`機構と、

遠隔`dbx`エンジンが作動しているかどうかを判定し、作動していない場合には、`dbxWrapperFactory`オブジェクトの機能を使用して`dbx`エンジンを作成し、遠隔ターゲット機能に接続する`IdentifyRemoteFunction`機構と、

`dbx`エンジンが互いに通信を行えるようにする多重`dbx`エンジン同期機構とを備えている`dbx`エンジン。

【請求項12】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法において、

`SPARCworks dbx`エンジンのものと同等な機能を有する前記標準`dbx`エンジンを設けるステップと、

コンピュータの制御の下で、前記標準`dbx`エンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「IDL」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）`dstep`機構を設けるステップとを備えているコンピュータ実行方法。

【請求項13】 コンピュータの制御の下で前記`dbx`エンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「IDL」）生成コード機構のどれを無視すべきかを決定する`remote surrogate code test`機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項12に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法。

4

【請求項14】 コンピュータの制御の下で前記`dbx`エンジンに、ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項12に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法。

【請求項15】 コンピュータの制御の下で前記`dbx`エンジンに、指定されたオブジェクトのホストIDおよびプロセスID（PID）を探し出す`GetImplementation`機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項12に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法。

【請求項16】 コンピュータの制御の下で前記`dbx`エンジンに、遠隔`dbx`エンジンが作動しているかどうかを判定し、作動していない場合には、`dbxWrapperFactory`オブジェクトの機能を使用して`dbx`エンジンを作成し、遠隔ターゲット機能に接続する`IdentifyRemoteFunction`機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項12に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法。

【請求項17】 コンピュータの制御の下で前記`dbx`エンジンに、`dbx`エンジンが互いに通信を行えるようにする多重`dbx`エンジン同期機構を追加する付加的なステップをさらに含んでいることを特徴とする、請求項12に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法。

【請求項18】 ローカル・ホスト・コンピュータおよび1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムで使用する標準`dbx`エンジンを修正するコンピュータ実行方法において、`SPARCworks dbx`エンジンのものと同等な機能を有する前記標準`dbx`エンジンを設けるステップと、

コンピュータの制御の下で、前記標準`dbx`エンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「IDL」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）

5

d s t e p 機構を設けるステップと、
 前記標準 d b x エンジンに、前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続する中間インタフェース定義言語（「IDL」）生成コード機構のどれを無視すべきかを決定する r e m o t e s u r r o g a t e c o d e t e s t 機構を設けるステップと、
 前記標準 d b x エンジンに、ローカル・ホスト・コンピュータのユーザが実際には遠隔ホスト・コンピュータで実施されている分散ターゲット・アプリケーション・システムの機能にブレークポイントをセットすることを可能とする遠隔ブレークポイント設定機構を設けるステップと、
 前記標準 d b x エンジンに、指定されたオブジェクトのホスト ID およびプロセス ID（PID）を探し出す G e t I m p l e m e n t a t i o n 機構を設けるステップと、
 前記標準 d b x エンジンに、遠隔 d b x エンジンが作動しているかどうかを判定し、作動していない場合には、d b x W r a p p e r F a c t o r y オブジェクトの機能を使用して d b x エンジンを作成し、遠隔ターゲット機能に接続する I d e n t i f y R e m o t e F u n c t i o n 機構を設けるステップと、
 前記標準 d b x エンジンに、d b x エンジンが互いに通信を行えるようにする多重 d b x エンジン同期機構を設けるステップとを備えているコンピュータ実行方法。
 【請求項 19】 ローカル・ホスト・コンピュータおよび 1 つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法において、
 ローカル・ホスト・コンピュータに、デバッガ GUI および 1 つまたは複数のデバッガ・エンジン（以下、「d b x エンジン」と呼ぶ）であって、前記デバッガ GUI が前記 d b x エンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えているデバッガ GUI および 1 つまたは複数の d b x エンジンを設けるステップと、
 メッセージを互いに送受信するために前記 d b x エンジンと前記デバッガ GUI が使用する通信機構を設けるステップと、
 前記の 1 つまたは複数の d b x エンジンの 1 つであり、前記ローカル・ホスト・コンピュータから離隔したホスト・コンピュータにあり、前記通信機構によって前記デバッガ GUI に接続されている遠隔 d b x エンジンを設けるステップとを備えており、前記遠隔 d b x エンジンが前記デバッガ GUI からの指示の下で、前記遠隔ホスト・コンピュータにある前記分散ターゲット・アプリケーション・システムの部分に接続して、前記遠隔ホスト・コンピュータにある前記分散ターゲット・アプリケー

6

ション・システムの前記部分をデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項 20】 ローカル・ホスト・コンピュータおよび 1 つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法において、

デバッガ GUI および 1 つまたは複数のデバッガ・エンジン（以下、「d b x エンジン」と呼ぶ）であって、前記デバッガ GUI が前記 d b x エンジンと通信し、かつ前記デバッガ・システムのユーザと通信するためのインタフェース機構を備えており、前記 d b x エンジンが前記ローカルおよび遠隔のコンピュータにあることができるデバッガ GUI および 1 つまたは複数の d b x エンジンを設けるステップと、

メッセージを互いに送受信するために前記 d b x エンジンと前記デバッガ GUI が使用する通信機構を設けるステップと、

遠隔ホスト・コンピュータにある前記ターゲット・アプリケーション・システムの一部をデバッグする際に使用される新しい d b x エンジンを前記遠隔ホスト・コンピュータに作成するために前記デバッガ GUI が使用する d b x W r a p p e r F a c t o r y 機構を設けるステップとを備えている分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項 21】 前記ターゲット・アプリケーション・システムのローカル部分と遠隔部分とを接続するが、前記ターゲット・アプリケーション・システム自体の一部ではない中間インタフェース定義言語（「IDL」）生成コード機構を無視する（すなわち、「ステップ・オーバーする」）d s t e p 機構を設けるステップをさらに含んでいることを特徴とする、請求項 20 に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項 22】 前記 d b x エンジンの 1 つが前記 d b x エンジンの他のものと、これらの d b x エンジンが異なるホスト・コンピュータにあるかどうかにかかわらず、通信することを可能とする第 1 の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項 20 に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項 23】 前記デバッガ GUI が前記 d b x エンジンの 1 つに、前記 d b x エンジンがどのホスト・コンピュータにあるかにかかわらず、フォーカスすることを可能とする第 2 の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項 20 に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行

方法。

【請求項24】 前記ユーザが、前記d b xエンジンがどのホスト・コンピュータにあるかにかかわらず、すべての活動d b xエンジンのリストを前記デバッガGUIから取得できるようにする第3の通信機構を設けるステップをさらに含んでいることを特徴とする、請求項20に記載の分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムを作成するコンピュータ実行方法。

【請求項25】 プロセスを作成し、分散コンピューティング環境において前記プロセスと通信する分散コンピューティング環境におけるコンピュータ実行方法において、

- a. クライアント・プロセスが要求されているサービスが遠隔であるかどうかを判定するステップと、
- b. 前記サービスが遠隔であると前記クライアント・プロセスが判定した場合に、前記サービスがおかれている遠隔ロケーションを決定するステップと、
- c. 前記クライアント・プロセスが前記遠隔ロケーションにおいて前記サービスを有しているサーバ・プロセスにメッセージを送って、前記サーバに前記クライアント・プロセスとの通信を確立するよう指示するステップと、
- d. 前記サーバ・プロセスが前記クライアント・プロセスと通信するインタフェース・プロセスを作成するステップと、
- e. 前記インタフェース・プロセスが前記サーバ・プロセスに接続するまで、前記サーバ・プロセスが操作を中断するステップと、
- f. 前記インタフェース・プロセスが前記サーバ・プロセスへの接続を試みるステップと、
- g. 前記インタフェース・プロセスが前記サーバ・プロセスに接続できない場合に、前記インタフェース・プロセスが前記クライアント・プロセスにアラートし、そうでない場合には、前記クライアント・プロセスが前記サーバ・プロセスに接続して、前記接続が成功した旨を前記クライアント・プロセスにアラートするステップと、
- h. 前記クライアント・プロセスが前記インタフェース・プロセスを介して前記サーバ・プロセスと通信するステップと

を備えている分散コンピューティング環境におけるコンピュータ実行方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は分散コンピューティング・システム、クライアント・サーバ・コンピューティングおよびオブジェクト指向プログラミングの分野に関する。具体的にいえば、本発明はプログラム開発者およびユーザに分散サーバ上にプログラムまたはオブジェクトを含んでいることのあるターゲット・アプリケーシ

ョンをデバッグする能力を与える方法および装置に関する。

【0002】

【従来の技術】コンピュータ・アプリケーション開発者が作成中のアプリケーションをデバッグできることは必須である。これはオブジェクト指向分散プロセッサ環境においては、きわめて困難な問題となっている。このような現代の環境はアプリケーション開発者以外の人間が開発したオブジェクトを呼び出すアプリケーションを含んでおり、かつアプリケーション開発者から遠隔の、当該開発者には未知のプロセッサで作動するオブジェクトのインプリメンテーションを含んでいる可能性もある。それにもかかわらず、アプリケーション開発者には、このような遠隔で、未知のプロセッサに常駐するアプリケーションの部分デバッグする方法がなければならない。

【0003】たとえば、分散アプリケーションは2つ以上の部分からなるアプリケーションである。これらの部分はクライアントおよびそのサーバと呼ばれることがしばしばある。分散アプリケーションは何年も前から存在しているものであり、それだけの年月の間、プログラム・アプリケーション開発者は分散アプリケーションをデバッグする問題を抱えていた。分散アプリケーションの典型的なデバッグ法は、デバッガの下でクライアントを起動し、サーバにある機能に達するまでクライアントをデバッグすることである。開発者が幸運であれば、サーバがすでに既知のホストで作動している。開発者は次いでサーバ・ホストへ進み、サーバ・プロセスを識別し、デバッガをこれに接続し、デバッグ・セッションを継続する。サーバがまだ作動していない場合には、開発者はサーバを作動させる方法を見つけだし、自分が行ったことで探しているバグが隠ぺいされないよう祈る他はなくなる。サーバが起動されたら、開発者はデバッガをこれに接続する。あるいは、サーバの起動に干渉して、何か興味深いことが起きる前にデバッガをサーバに接続する方法を、開発者が見つけださなければならない。この方法はエラーを起こしやすいものであり、手間がかかり、またしばしばわかりにくく、単調なものである。

【0004】オブジェクト指向システムにおいて、オブジェクトはデータと、データを処理するために呼び出すことのできる操作（オペレーション）とからなる構成要素である。操作（「メソッド」とも呼ばれる）はオブジェクトにコールを送ることによってオブジェクトに呼び出される。各オブジェクトはオブジェクト・タイプを有しており、このタイプはそのタイプのオブジェクトで行うことのできる操作を定義している。あるオブジェクト・タイプは他のオブジェクト・タイプについて定義され、実施されたオブジェクト操作を継承することができる。オブジェクト指向設計およびプログラミング技法の詳細については、Bertrand Meyerの「O

bject-oriented Software Construction」、Prentice-Hall、1988年を参照されたい。

【0005】クライアント・サーバ・コンピューティングにおいては、通常、コンピュータを接続しているネットワークを介して互いに通信を行える1組のコンピュータがある。これらのコンピュータの中には、他のコンピュータに対するサービスまたは機能のプロバイダとして働くものがある。サービスまたは機能のプロバイダは「サーバ」と呼ばれ、サービスまたは機能の消費者は「クライアント」と呼ばれる。クライアント・サーバ・モデルは、同一のコンピュータで動作している別個のプログラムが何らかの保護機構を介して互いに通信を行い、機能のプロバイダと消費者として働いているというケースに一般化することもできる。

【0006】クライアント・サーバ・モデルに基づくオブジェクト指向分散システムにおいては、オブジェクト指向インタフェースをクライアントに与えるサーバがある。これらのサーバは、データと、このタイプのオブジェクトで認められる操作にしたがってデータを処理する関連ソフトウェアとからなるオブジェクトを、サポートしている。クライアントはこれらのオブジェクトに対するアクセスを取得し、コールをサーバに伝送することによってこれらのオブジェクトでコールを実行することができる。サーバ側では、これらのコールはオブジェクトに関連するソフトウェアによって実行される。これらのコールの結果はクライアントへ返送される。

【0007】従来技術のデバッグによる他の基本的な問題は、現代の分散オブジェクト・システムで実施されたアプリケーションをデバッグしようとするときに生じる。オブジェクト管理グループ（「OMG」）によって一般的に規定されたタイプの分散オブジェクト・システムを考える。OMGは分散オブジェクト・システムのいくつかの仕様およびプロトコルに合意した500社以上の会社の団体である。このシステムの基本仕様は「The Common Object Request Broker: Architecture and Specification」（「CORBA」ともいう）という題名の1993年12月29日付のOMG Document No. 93.xx.yy Revision 1.2に記載されている（この基本仕様はここでの引用によって本明細書の一部となる）。CORBA準拠システムは既存のオブジェクトによってアプリケーションを構築することに備えたものである。このようなアプリケーションはオブジェクトの作成を要求し、かつそのオブジェクトで操作を行うことができる。オブジェクトの作成およびその上での操作はこれらのオブジェクトのサーバによって行われる。このようなアプリケーションはオブジェクトを作成しようとした場合、そのオブジェクトの「ファクトリ」と呼ばれるサーバを探し出すロ

ケータ機構を透過的に利用する。同様に、このようなアプリケーションが既存のオブジェクトを有している場合には、ロケータ機構を透過的に利用して、そのオブジェクトでの操作を行うことのできるサーバを探し出す。

【0008】

【発明が解決しようとする課題】このようなCORBA準拠システムにおいては、アプリケーション・プログラマがそのオブジェクトのサーバが作動している場所についての知識なしにオブジェクトを使用するのを可能とする相当な量の機構（メカニズム）が各オブジェクトの背後にある。クライアントの開発者がサーバの開発者でもあるという特別な状況では、サーバがどこで作動することになるのか、またその名前が何かということを経営者が知ることができるよう構成をとることができる。しかしながら、一般に、CORBA準拠システムのアプリケーション開発者は自分のオブジェクトに関連したサーバを探し出すことができない。それ故、オブジェクトが同一のプロセスに配置されているのか、遠隔のプロセスに配置されているのかにかかわらず、またオブジェクトがどこにあるのかにかかわらず、アプリケーションが使用するオブジェクトのデバッグをサポートする必要がある。さらに、このデバッグ手順が開発者に「単一プロセス」であると思わせ、熟知したデバッグ・パラダイムを使用して大規模な分散アプリケーションのデバッグを行えるようにすることが好ましい。

【0009】いくつかの従来技術のデバッグの主な欠点は、これらがデータおよびプロセスをサポートするのに、通常は関連するコンパイラが生成する付加的なデータを含んでいる大きなオーバーヘッドを必要とすることである。したがって、遠隔デバッグの好ましい実施の形態は、オブジェクトの実装者がオブジェクトを「デバッグ可能」にするのに、サーバをコンパイルして、記号情報を生成する以外の特別なことを何もする必要のないようにする必要がある（-gコンパイラ・オプションを使用したCおよびC++の場合）。しかしながら、関連するオーバーヘッドが細分化されたオブジェクトのサイズおよびパフォーマンスに影響を及ぼさないように、サーバまたはサーバントのいずれかでの付加的な挙動上の抽象化やデータの抽象化を必要としてはならない。同様に、従来技術のデバッグ・システムの他の制限は、これらが特定のタイプのターゲット・アプリケーションおよび／または特定のコンパイラ言語にリンクされていることである。分散デバッグが実装言語と無関係にアプリケーションをデバッグできることが望ましい。すなわち、好ましい分散デバッグはこれが作動するサーバおよびオブジェクトの種類についていかなる想定もしてはならない。CORBA仕様は各種のオブジェクトの実装（インプリメンテーション）をサービスするのに使用される各種の「オブジェクト・アダプタ」を記述している。望ましい分散デバッグは、これが作動するサーバやオブジェ

クトの種類について実際にいかなる想定もしないのであれば、「オブジェクト・アダプタ」に無関係な態様で作動しなければならない。さらに、CORBA準拠システムのインプリメンテーションがシステムの作動を容易とするために使用することのあるボイラプレート・コードを無視できる分散デバッガを得ることが望ましい。「ボイラプレート」コードとは開発環境によって作成され、開発者にはわからない非プログラマ生成コードをいう。分散デバッガは開発者がシステムを実装した抽象化と同じ機能レベルでシステムをデバッグすることを可能とするものでなければならない。

【0010】「doeDebugger」と呼ぶ本発明の分散デバッガはシームレスで、オーバーヘッドが少なく、わずらわしくない態様で分散デバッグを達成し、これによって開発者が「単一プロセス」のアプリケーションをデバッグしているのだと思いこんで、分散オブジェクト指向アプリケーションをデバッグすることを可能とする装置および方法を提供する。

【0011】

【課題を解決するための手段】クライアント・アプリケーションがローカル・ホストでデバッガを使用できるとともに、オブジェクトを含んでいるアプリケーションおよび未知の遠隔ホスト・コンピュータで作動するオブジェクト・インプリメンテーションをシームレスにデバッグできる装置および方法を開示する。

【0012】一部がローカル・ホスト・コンピュータにあり、一部が1つまたは複数の遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、ローカル・ホスト・コンピュータまたは遠隔ホスト・コンピュータにおかれているデバッガGUIおよび1つまたは複数のdbxエンジンと、dbxエンジンとデバッガGUIが互いに会話をを行うために使用する通信機構とを有している分散デバッガ・システムを開示する。

【0013】特許請求の範囲に記載する本発明の他の態様は、一部がローカル・ホスト・コンピュータにあり、一部が遠隔ホスト・コンピュータにある分散ターゲット・アプリケーション・システムをデバッグする分散デバッガ・システムにおいて、ローカル・ホスト・コンピュータまたは遠隔ホスト・コンピュータにおかれているデバッガGUIおよび1つまたは複数のdbxエンジンと、dbxエンジンとデバッガGUIが互いに会話をを行うために使用する通信機構とを有しており、さらに希望する遠隔デバッグ・サポートをもたらすため、必要に応じ、遠隔ホスト・コンピュータに新しいdbxエンジンを作成するのにデバッガGUIが使用するdbxWrapperFactory機構を有している分散デバッガ・システムを含む。

【0014】本発明はこれらの操作を行う装置にも関する。この装置は必要とされる目的に合わせて特に構成さ

れたものであっても、コンピュータに記憶されているコンピュータ・プログラムによって選択的に起動されるか、あるいは再構成される汎用コンピュータを備えたものであってもよい。本明細書に記載する作動は特定のコンピュータまたはその他の装置に本質的に関連したものではない。詳細に言えば、各種の汎用マシンを本発明の教示にしたがって作成されたプログラムとともに使用してもよいし、あるいは必要とされる方法のステップを行うための専用マシンの方が便利なこともある。各種のこれらのマシンに必要な構造は以下の説明から明かとなるう。

【0015】特許請求の範囲に記載する本発明の他の態様は、ローカル・ホスト・コンピュータと1つまたは複数の遠隔ホスト・コンピュータ装置におかれる分散・ターゲット・アプリケーション・システムのデバッグ時に分散デバッガ・システムで使用するdbxエンジンにおいて、非プログラマ生成コードを無視するdstep機構と、このような非プログラマ生成コード(IDL生成コードともいう)を識別するテスト機構と、ターゲット・アプリケーション・システムのセクションに遠隔ブレークポイントをセットする機構と、呼び出されたオブジェクトを実施するサーバのホストIDおよびプロセスID(pid)を識別するGetImplementation機構(「サーバ検索」機構ともいう)と、dbxエンジンが互いに通信を行えるようにする多重dbxエンジン同期機構とからなるdbxエンジンを含んでいる。

【0016】本願の特許請求の範囲には、上記の特性を有するdbxエンジンを作成する方法、ならびに上記のような分散デバッガ・システムを作成する方法も記載する。

【0017】本発明の目的、特徴および利点は以下の説明から明かとなるう。

【0018】

【発明の実施の形態】表記および用語以下の詳細な説明はコンピュータまたはコンピュータのネットワークで実行されるプログラム・プロシージャによって表される。これらのプロシージャの記述および表現は当分野の技術者が自分達の作業の主題を他の技術者に最も効率よく伝えるために使用する手段である。

【0019】プロシージャは本明細書において、また一般に、希望する結果にいたる一貫した一連のステップであると考えられる。これらのステップは物理量の物理的取扱いを必要とするものである。必ずしもそうとは限らないが、一般に、これらの量は記憶、転送、組合せ、比較、あるいは処理を行える電気信号または磁気信号の形態をしている。主として一般的に使用するため、これらの信号をビット、値、要素、記号、文字、項、数値などと呼ぶのが便利なこともある。ただし、これらや類似した用語が適切な物理量と関連しており、これらの量に適

用された都合のよいラベルに過ぎないことに留意すべきである。

【0020】さらに、実行される処理は操作員が行う知的な操作と一般に関連づけられている乗算や比較などの用語で呼ばれることがしばしばある。本発明の一部を形成する本明細書に記載する操作のいずれにおいても、操作員のこのような能力は必要ではなく、またほとんどの場合、望ましくない。操作はマシン操作である。本発明の操作を行うのに有用なマシンとしては、汎用デジタル・コンピュータまたは類似の装置がある。

【0021】本発明はこれらの操作を行う装置にも関する。この装置は必要とされる目的に合わせて特に構成されたものであっても、コンピュータに記憶されているコンピュータ・プログラムによって選択的に活動化されるか、あるいは再構成される汎用コンピュータを備えたものであってもよい。本明細書に記載する作動は特定のコンピュータまたはその他の装置に本質的に関連したものではない。各種の汎用マシンを本発明の教示にしたがって作成されたプログラムとともに使用してもよいし、あるいは必要とされる方法のステップを行うための専用マシンの方が便利なることもある。各種のこれらのマシンに必要な構造は以下の説明から明かとなる。

【0022】以下の開示はアプリケーションのプログラマ／開発者が1台のホストマシンのところにおり、開発されるアプリケーションがプログラマ／開発者には未知の他のホストマシンにおかれていることのあるオブジェクトおよびオブジェクト・インプリメンテーションを利用する分散コンピュータ・アプリケーションをデバッグするシステムおよび方法に関する。このシステムおよび方法は広く分散したオブジェクト指向クライアント・サーバ・システムにおけるオブジェクトの使用に関連づけられた新しいアプリケーションをデバッグを試みる際に遭遇する問題の解決策を提供する。分散オブジェクトはこれらが要求を他のオブジェクトへ送るのか、あるいはクライアントからの要求に回答しているのかに応じてオブジェクト・クライアントにも、オブジェクト・サーバにもなり得る。分散オブジェクト環境において、要求および応答はオブジェクトの場所および状況を認識しているオブジェクト・リクエスト・ブローカ（ORB）によって行われる。ORBを実施するのに適している1つのアーキテクチャが、共通オブジェクト・リクエスト・ブローカ・アーキテクチャ（CORBA）仕様によって与えられる。任意の関連した状況で使用できるものであるが、記載されているインプリメンテーションはSun Microsystems, Inc. の分散オブジェクト環境（「DOE」）の拡張である。しかしながら、本開示で記載するプロセスおよびシステムを理解し、実施するのに、当分野の技術者がDOEシステムの詳細な知識を必要とすることはない。

【0023】本発明はプログラマ／開発者のアプリケー

ションが呼び出したオブジェクトが遠隔で実施されていることを、プログラマ／開発者が無視できるようにし、また遠隔インプリメンテーションであることによる特別なことを、プログラマ／開発者が行うことを要求せず、しかも過度のオーバヘッド負荷を課することがなく、安全であり、プログラマ／開発者がシステムを実施したのと同じ抽象化の機能レベルで、プログラマ／開発者がシステムをデバッグするのを可能とするdoeDebuggerを作成し、使用するシステムおよび方法を開示する。他のインプリメンテーションは若干の言語非依存性ももたらす。

【0024】1. 定義

本明細書では、「分散オブジェクト」または「オブジェクト」という用語は、オブジェクトと関連づけられた定義済みのインタフェースを介した操作によって処理できるコードおよびデータのカプセル化されたパッケージをいう。それ故、当分野の技術者にとって、分散オブジェクトは従来のプログラミング・オブジェクトを定義している基本特性を含むものとみなされる。しかしながら、分散オブジェクトは従来のプログラミング・オブジェクトとは、2つの重要な特徴を含んでいることによって相違している。まず、分散オブジェクトは多言語のものである。分散オブジェクトのインタフェースは各種の異なるプログラミング言語にマップできるインタフェース定義言語を使用して定義される。このようなインタフェース定義言語の1つがOMG IDLである。第2に、分散オブジェクトは場所に依存しない。すなわち、分散オブジェクトはネットワーク内のどこにでもおける。これは単一のアドレス・スペース、すなわち「クライアント」のアドレス・スペースに存在しているのが典型的な従来のプログラミング・オブジェクトと明確に異なる場所である。分散オブジェクトは他のオブジェクトに要求を送っているのか、あるいは他のオブジェクトからの要求に回答しているのかに応じてオブジェクト・クライアントにも、オブジェクト・サーバにもなり得る。要求および応答はオブジェクトの場所および状況を認識しているオブジェクト・リクエスト・ブローカ（ORB）によって行われる。

【0025】「分散オブジェクト・システム」ないし「分散オブジェクト・オペレーティング環境」とは、ORBを介して通信を行う分散オブジェクトからなるシステムをいう。

【0026】「オブジェクト・リファレンス」ないし「object ref」とは、他のオブジェクトへのポインタを含んでいるデータ構造（従来のプログラミング言語オブジェクトでもよい）をいう。オブジェクト・リファレンスの作成および定義については、当分野の技術者は熟知しているであろう。

【0027】本明細書で定義する「クライアント」とは、オブジェクトに要求を送るエンティティである。こ

のモデルにおいて、オブジェクトは、サービスを与えるものであり、「サーバ・オブジェクト」または「ターゲット・オブジェクト」または「オブジェクト・インプリメンテーション」と呼ばれる。したがって、クライアントはサーバから操作、ないしインプリメンテーションを呼び出す。場合によっては、クライアントはそれ自体でオブジェクトである。分散オブジェクト環境において、クライアントはインプリメンテーション・プログラミング言語についての知識を持っていなくてもよく、あるいは、このようなオブジェクトの多言語性の要件により、インプリメンテーションがクライアントのプログラミング言語の知識を持っていなくてもよい。分散オブジェクト・オペレーティング環境のクライアントおよびサーバに必要なのは、インタフェース定義言語によって通信を行うことだけである。上述したように、クライアントによるサーバに対する要求およびクライアントに対するサーバの応答は、ORBによって処理される。指摘しておきたいのは、クライアントとサーバが同一のホスト・コンピュータまたは2台の異なるホスト・コンピュータの同一のプロセスに存在できることである。

【0028】「オブジェクト・インタフェース」はオブジェクトがもたらす操作（オペレーション）、属性（アトリビュート）および例外事項（エクセプション）の仕様である。分散オブジェクトのオブジェクト・インタフェースは許可されているインタフェース定義言語（IDL）を使用して作成することが好ましい。上記したように、オブジェクトはそのインタフェースを介してトランザクションを行う。したがって、インタフェースを使用することによって、トランザクションにおけるオブジェクトのメソッドおよびデータを定義するのに使用されたプログラミング言語をクライアントが認識している必要がなくなる。

【0029】情報のパケットを「マーシャル（marshal）」するとは、共用メモリ通信チャネル、あるいはネットワーク通信回線のいずれかによってこの情報を転送する準備を行うことである。これはしばしば、使用されている通信プロトコルにしたがって特定のフォーマットでデータを編成することをいう。

【0030】情報のパケットを「アンマーシャル（unmarshal）」するとは、本質的にマーシャル手順を逆にすることであり、該当する環境で有意のフォーマットでデータを作成することである。

【0031】「ToolTalk」とは、本願出願人Sun Microsystems, Inc. の子会社であるSunSoftが開発し、提供している通信システムである。「ToolTalk」はあるアプリケーションが作成したメッセージを、そのメッセージを受け取ることを要求している他のものへ送出する通信サービスを提供する。「ToolTalk」によって、独立したアプリケーションが他のアプリケーションと、相互の直接

的な知識を要することなく、通信を行うことが可能となる。送信者は「ToolTalk」メッセージを特定のプロセス、何らかの関心のあるプロセス、オブジェクト、あるいはオブジェクト・タイプにアドレス指定することができる。「ToolTalk」の詳細については「The ToolTalk Service: An Inter-Operability Solution, (ISBN 013-088717-X)」という題名のSunSoft Press/PTR Prentice Hallの書籍に記載されている。

【0032】11. 作動環境

本発明が使用される環境には、汎用コンピュータ、ワークステーションあるいはパーソナル・コンピュータがさまざまなタイプの通信リンクを介して、クライアント・サーバ構成で接続されており、プログラムやデータが多くがオブジェクトの形式で、システムの各種のメンバによって、システムの他のメンバによる実効およびアクセスのために利用できる汎用分散コンピューティング・システムが含まれている。汎用ワークステーション・コンピュータの要素のいくつかを図1に示す。図において、入出力（「I/O」）部2、中央処理装置（「CPU」）3およびメモリ部4を有するコンピュータ1が示されている。I/O部2はキーボード5、表示装置6、ディスク記憶装置9、およびCD-ROM駆動装置7に接続されている。CD-ROM装置7は、通常、プログラム10やデータを含んでいるCD-ROM媒体8を読み取ることができる。図2は典型的なマルチプロセッサ分散コンピュータ・システムを示しており、個々のコンピュータ20、22および24が通信リンク26を介して互いに接続され、かつ恐らくは共用メモリ装置28に接続されている。図3は典型的なオブジェクト指向クライアント・サーバ構成を示しており、ユーザ30が第1のコンピュータ32でクライアント・アプリケーション34を起動できる。クライアント・アプリケーション34はコール40をオブジェクト・リファレンス36に出し、このオブジェクト・リファレンスは第2のコンピュータ（サーバ）50のオブジェクトのインプリメンテーション（「ターゲット・オブジェクト」ともいう）46をポイントする。コール40は通信制御機構38に渡され、この機構38はコールをオブジェクト・インプリメンテーション46が配置されているサーバ50に送る。このオブジェクト・インプリメンテーション46は最初にオブジェクト・リファレンス36を作成し、ユーザに利用できるようにする。コールの処理の終了時に、オブジェクト・インプリメンテーション46はメッセージまたは希望した操作の結果を通信リンク42を介して起点クライアント・アプリケーション34に返す。このクライアント・サーバ・モデルは、通信機構の機能がオペレーティング・システム（図4の62）によって行われる単一プロセッサ装置でも機能する。

【0033】111. 分散デバッガー作成方法
個々で、図5を参照すると、SPARCworksデバッガー・システムが示されている。SPARCworksデバッガー（以下、「Debugger」と呼ぶ）はSun Microsystems, Inc. 製のSPARCworksツールセットの組み込みコンポーネントである。なお、SPARCworksツールセットはAnalyzer、dbxエンジン、FileMergeツール、Make tool、Manager、および、SourceBrowserを含む。Debuggerの詳細については、SunSoftが1994年8月にPart No. 801-7105-10として刊行した「Debugging a Program」という題名の文献に記載されている（これは引用により本明細書の一部とする）。

【0034】ここで、図5を参照すると、Debuggerはdbxエンジン76とインタフェースする洗練されたウィンドウ・ベースのツール72（Debugger・グラフィカル・ユーザ・インタフェース（GUI））を含んでいる。dbxエンジン76は対話式のライン指向ソースレベル記号デバッガーである。dbxエンジン76はターゲット・プログラムがクラッシュした場所を決定し、変数と式の値を調べ、コードにブレークポイント78をセットし、ターゲット・プログラムを実行し、トレースすることを可能とする。プログラムの実行中に、dbxエンジン76はターゲット・プログラムの挙動に関する詳細な情報を取得し、ToolTalk通信プロトコル74によってDebugger GUI72にこの情報を供給する。dbxエンジン76はコンパイラがコンパイラ・オプション-gを使用して生成したデバッギング情報に依存して、ターゲット・プロセスの状態を検査する。現行のSun Microsystems, Inc. のオペレーティング・システム環境であるSolaris 2.xのデフォルトでは、各プログラム・モジュールのデバッギング情報はモジュールの0ファイルに格納される。Solaris 2.xにおける好ましい実施の形態において、dbxエンジン76は必要に応じ、各モジュールに関する情報を読み込む。「ブレークポイント設定」78の機能に加えて、dbxエンジン76は「単一ステップ」80の機能、なら

びに上記の参照文献「Debugging a Program」に詳細が記載されている多くのその他の機能82を有している。「ステップ」80機能はプログラマー／開発者がターゲット・プログラム・コードをソース言語レベルまたはマシン語レベルのいずれかで1行ずつ単一のステップで実行し、ファンクションコールをステップ「over」またはステップ「into」し、ファンクション・コールをステップ「up」またはステップ「out」してコール側ファンクション行に到達すること（ただし、コール後に）を可能とする。ブレークポイ

ント78のコマンドには3つのタイプがある。

【0035】（1）stopタイプのブレークポイント： ターゲット・プログラムがstopコマンドを使用して作成されたブレークポイントに到達した場合、プログラムは停止し、任意選択で1つまたは複数のデバッグ・コマンドを発行する。ターゲット・プログラムをレジュームするには、他のデバッグ・コマンドを発行しなければならない。

【0036】（2）whenタイプのブレークポイント： ターゲット・プログラムが停止し、dbxエンジンが1つまたは複数のデバッグ・コマンドを発行し、その後ターゲット・プログラムが継続する。

【0037】（3）traceタイプのブレークポイント： ターゲット・プログラムが停止し、イベント固有のtrace行が排除され、その後プログラムが継続する。

【0038】非分散システムにおける、Debuggerの典型的な構成を図6に示す。図において、ToolTalk通信リンク96によってdbxエンジン98に接続されたデバッガーGUI94を含んでいるホストマシン92が示されており、dbxエンジン98はクライアント（ターゲット・プログラム）100にリンクされており、クライアントはさらに付加的なターゲット・アプリケーション・コード（サーバ）102に接続されている。分散システムにおいて、プログラマー／開発者は図7に示したような状態に直面する。図7は複数のホスト上にある複数のクライアントと複数のサーバを示している。赤ホスト112にあるDebugger114を使用して、クライアント116をデバッグするプログラマー／開発者には、クライアント116が赤ホスト112のサーバ118または青ホスト122のサーバ124によって実行できるオブジェクトで操作を行っており、クライアント116の開発者としてのプログラマーにはコールの実行にどのサーバが使用されるのかわからないということが判明する。さらに、クライアント116によって使用されるどのサーバも、白ホスト130のクライアント2128が使用できる。DOE（本発明の好ましい実施の形態）などのCORBA準拠の分散システムでは、かなりの量の機構（メカニズム）が各オブジェクトの背後にあり、この機構によって、アプリケーション・プログラマーがオブジェクトを、そのオブジェクトのサーバがどこで作動しているのかの知識を必要とせず、使用できるようになる。このことはこのような状況の下では、分散アプリケーションのデバッグ作業の重大な障害となる。さらに、すべてのDOEサーバはマルチスレッド・サーバであり、サーバが異なるクライアントからの複数の要求に同時に応えることができる。したがって、本発明のdoeDebuggerはDOE分散環境で生じるデバッグ上の問題の多くに対する解決策となる。

【0039】本発明のdoeDebuggerを作成するのに必要なDebuggerの改変について説明する前に、オブジェクトがCORBA準拠環境で互いに通信するのを可能とするためにDOEが使用するボイラプレート機構コードについて簡単に触れておくことが有用であろう。図8を参照すると、ユーザ・コードと基礎DOE機構の間の分割が示されている。

【0040】図8において、ローカル・ホスト142で、ユーザ・コードが機能、例えば、機能「foo」144を呼び出す。DOEシステムは機能fooとのインタフェースをもたらし生成コード148を備えており、生成コード148はメッセージ機能150を呼び出して、適切なメッセージを作成し、これをサーバ152に送る。これらのメッセージ154は該当するホスト156に到達し、サーバ側メッセージ・システム158によって受け取られ、このメッセージ・システムはメッセージをDOEシステム160が生成したコードであるサーバ側スタブに渡す。DOEシステム160は次いで機能foo162のインプリメンテーション・コードを呼び出し、この機能のインプリメンテーションのユーザ・コードがコール164を処理する。アプリケーション・プログラマはdoeサーバによってもたらされるオブジェクトを利用するコード144を書く。アプリケーション・プログラマはインタフェース定義言語（IDL）を使用してオブジェクトに対するインタフェースを定義する。IDLはクライアント側およびサーバ側のライブラリにコンパイルされ、これらのライブラリはクライアントにあるオブジェクトに対する操作をサーバで行うことを可能とする。クライアント側ライブラリは操作をサーバに対する要求154に変換するスタブ機能148、150からなっている。サーバ側ライブラリ160、162はクライアントからの要求をその操作を実施するユーザ提供機能164に変換する。

【0041】たとえば、図8において、ユーザは機能fooをIDLに定義している。IDLコンパイラはスタブ機能foo148、150を生成し、サーバに送られるメッセージを作成させる（152）。サーバ側では、IDLが生成したコード160、162がクライアントからのメッセージを取り入れ、これをプログラマによって与えられる、fooの機能を実現する、機能fooに対するコールに変換する（164）。この基礎コード148、150、152、158、160および162はすべてプログラマ／開発者がデバッグしようと思っておらず、また正規のデバッグ・セッションにおいてデバッグに調べさせようと思っていないコードである。したがって、分散デバッグ・セッションがプログラマ／開発差に関する限り、この基礎コードをすべて無視できるようにすることが望ましい。

【0042】図9はSPARCworksデバッガを本発明のdoeDebuggerに変換するのに必要な改

変および拡張を示す。実際の拡張は共用ライブラリ「libdoeDebugger.so」204にパッケージされる。必要とされる基本的な拡張には次のようなものがある。

【0043】「dstep」コマンド206

「remote surrogate code test」機構208

「Remote Breakpoint」設定機構210

「GetImplementation」機構212

「IdentifyRemoteFunction」機構214

「multiple dbx engine synchronizer」機構216

【0044】これらの拡張に加えて、doeDebuggerの作動をサポートするためのDebugger GUIおよびdbxエンジンの改変には、次のものがある。

【0045】あるdbxエンジンが他のdbxエンジンと通信を行うための通信機能

デバッガGUIからすべての活動dbxエンジンのリストを取得する機能

【0046】これらの改変の好ましい実施の形態を総括的に説明するが、当分野の技術者にはこれらの機能および特徴が最も適切なものである部分に対するハードウェア機構および装置を含む多くの形態で実現できることが認識されよう。

【0047】「dstep」コマンドは分散システムのどこに所与の機能のインプリメンテーションが実際にあるのかにかかわらず、所与の機能のインプリメンテーションにシームレスに取りかかるために、プログラマ／開発者によって使用される。「dstep」はまず正規のdbx「step」コマンドを発行することによって作動する。標準の「step」コマンドはデバッグされるプロセス（デバッガー）の実行をデバッガーの現行のソース・ラインから次のソース・ラインまで継続する。現行ラインが機能呼び出す点になると、次のソース・ラインが呼び出される機能の最初のソース・ラインになる。stepコマンドのセマンティックスを分散アプリケーションのデバッグに拡張するため、拡張されたstepが図8のクライアントのfooなどの機能に入ると（点A 146）、実行がサーバのfooのインプリメンテーションの最初のラインで停止する（点B 166）。拡張stepコマンド（「dstep」コマンドと呼ぶ）は以下の2つの状況以外では、標準のstepコマンドと同じ作動を行う。

【0048】「dstep」コマンドがサーバでの呼出しをもたらし機能の呼出し時に実行された場合、次のソース・ラインがサーバで呼び出された機能の最初のラインになる。

【0049】「dstep」コマンドがサーバで呼び出された機能からの復帰時に実行された場合、次のソース・ラインは遠隔呼出しをもたらしたファンクション呼出し後にクライアントのソース・ラインとなる。DOEクライアントから見ると、「dstep」コマンドはユーザがIDL生成コードから戻ってきたときに、このコマンドの特別な機能を開始する。DOEサーバから見ると、「dstep」コマンドはユーザがIDL生成コードから戻ってきたときに、このコマンドの特別な機能を開始する。

【0050】doeDebuggerを透過的に遮断することも必要である。遠隔デバ깅をサポートするのに何を行ったのかをユーザが知らないのであるから、ユーザがこれを取り消すことを期待すべきではない。

【0051】「dstep」コマンドが実行された後、doeDebuggerは現行機能が「remote surrogate code」であるかどうかを判定しようとする。「remote surrogate code」はIDL操作の遠隔呼出しを担当するコードである（すなわち、図8の項目148、150）。DOEシステムでは現在、「remote surrogate code」はすべてIDLコンパイラによって生成されている。

【0052】クライアント側では、doeDebuggerがIDL生成コードを下がっていていることを認識する必要がある。これを達成するため、DOEはIDL生成コードの最初の層で使用される変数に特別な名前をつける。この変数が存在することが「dstep」機能を開始するトリガとして役立つ。

【0053】考えられるとおり、呼び出されるクライアント側の機能の名前は、その機能を実施するサーバ側の機能の名前と同じである。図8の例において、このことはプログラマがクライアント側で機能「foo」をコールし、自分がサーバ側で書いた機能「foo」を得ることを期待することに過ぎない。サーバは多くのクライアントにサービスを行えることがあり、したがって、その機能「foo」の多くの呼出しがサーバで生じることもある。サーバでの関連する呼出しは、サーバのどのスレッドがデバグ対象クライアントのファンクション・コールに対応しているかを判定することによって特定される。2つの機能（クライアント側のものと、サーバ側のもの）をDOEのメッセージ通過層に追加して、サーバの特定のスレッドの識別を援助する。

【0054】あるオブジェクトに対するサーバを探し出すために、「find server」機能がDOEベース・クラスに追加されている。これは「GetImplementation」という機能を実行する。この機能は任意のDOEオブジェクトによって呼び出すことができ、サーバのホストidおよびサーバのプロセスid(pid)を返すこととなる。doeDebugge

rはクライアントの「find server」機能を呼び出す。この機能はクライアントが使用しているオブジェクトの一部でなければならないから、doeDebuggerに組み込まれた機能ではない。「find server」機能が呼び出された時点で、サーバが作動していなければ、サーバは起動される。

【0055】上述のように、特別な変数が存在していることが、機能に取りかかるときに「dstep」機能を呼び出さなければならないことを通知する「trigger」として使用される。「dstep」がサーバから戻る時期を通知するのには、次の2つの理由で、同様な「trigger」は使用されない。

【0056】トリガ機構が復帰時に利用できなかった。および

【0057】呼び出された復帰がすでにわかっているコードを通して戻り（すなわち、このコードによってサーバに入った）、この知識を使って、復帰をトリガできた。

【0058】サーバに初めて入ったとき、機能「foo」（図8のIDLが生成した機能）についてのユーザのインプリメンテーションを直接呼び出す機能のスタック・ポインタ（復帰トリガと呼ぶ）がセーブされる。サーバからの復帰をチェックするとき、現行機能のスタック・ポインタを復帰トリガに関してチェックする。一致すれば、サーバから抜けることを継続する。

【0059】doeDebuggerが初めてサーバに入るときには、新しいdbxエンジンをサーバで起動しなければならない。このプロセスは「Identify Remote Function」プロセスの一部であり、遠隔dbxエンジンを識別し、dbxWrapperFactoryオブジェクトを使用して遠隔dbxエンジンを起動／作成する拡張機能として追加されたものである。これを行う方法を図14を参照して説明する。図14はデバグGUI502、dbxエンジン504、ヘルパ・プロセス506、クライアント側ラッパ・サーバ510およびクライアント508を有するローカル・ホスト520を示している。また、dbxエンジン512、ヘルパ・プロセス514、サーバ側ラッパ・サーバ518およびサーバ（呼び出された機能のインプリメンテーション）516を含んでいる遠隔ホスト522も示されている。デバグGUIへの接続およびサーバへの接続を含む新しいdbxエンジンの初期化は、新しいdbxエンジンを作成するコールを行うクライアント側dbxエンジン504によって達成される。

【0060】ローカル・ホスト520のdbxエンジン504はローカル・ホスト520のヘルパ・プロセス506を介したローカル・ホスト520のラッパ・サーバ510への要求によって、遠隔ホスト522にdbxエンジン512を作成する。ヘルパ・プロセス506は、ラッパ・サーバ510と通信を行うDOEアプリケーション

ョンである。dbxエンジン自体がマルチスレッド安全(MTセーフ)ではなく、DOEアプリケーションに作成することができない(すべてのDOEアプリケーションは元来マルチスレッド化されている)ため、これが必要となる。dbxエンジンはヘルパ・プロセスを介してラッパ・サーバのサービスにアクセスする。ローカル・ホスト520のラッパ・サーバ510は遠隔ホスト522のラッパ・サーバにメッセージを送ってdbxエンジンを遠隔サーバ522に作成し、サーバ516に接続するよう命令することを要求する。遠隔ホスト522にdbxエンジンを作成するようというローカル・ホスト520のdbxエンジン504による要求は、遠隔ホスト522のdbxエンジン512が完全に起動されるまで完了しない。遠隔ホスト522のラッパ・サーバは分岐し、遠隔ホスト522の新しいdbxエンジン512を実行し、dbxエンジン512が終了するか、あるいはdbxエンジン512が完全に起動されたことを示すメッセージを(ラッパ・サーバ518に)送るかするのを待つ。遠隔ホスト522のラッパ・サーバ518は2つのスレッドを作成する。一方のスレッドは分岐した子(dbxエンジン512)が終了するのを待機する。他方のスレッドは完全に起動されたdbxエンジン512からのメッセージを待つ。これらのスレッドの一方がレジュームされると、子のスレッドは他方のスレッドを破壊し、適切な応答をローカル・ホスト520のラッパ・サーバ510に戻し、このラッパ・サーバは遠隔ホスト522でのdbxエンジン512の作成要求を完了し、適切な状況値(作成成功または失敗)をローカル・ホストのdbxエンジン504に返す。

【0061】サーバの新しいdbxエンジン512が起動した(上述のようにして)後、クライアントのdbxエンジン504は新しいdbxエンジン512にメッセージを送って、サーバ516に適切なブレークポイントをセットする。クライアントのdbxエンジン504はサーバのdbxエンジン512が完全に起動するまで(すなわち、デバッグGUIとの接続を含む初期化を終わり、サーバに接続するまで)、このdbxエンジン512にブレークポイント・メッセージを送ることができない。

【0062】競合状態(レイス・コンディション)の可能性を回避するのに必要なdoeDebuggerの作動に関する付加的な事項を、ある程度詳細に説明する。

【0063】「dstep」中に行われるステップの記述はコマンドをサーバに接続されたdbxエンジンに送って、サーバにブレークポイントをセットすることをいう。コマンドはサーバの適正なスレッドにブレークポイントをセットするのに十分な情報を含んでいる。ステップがトリガ機能に入ると、トリガ機能のコールで生じたコールを処理するサーバのスレッドを一意に識別できる情報はまだ存在していない。実際の機構はトリガ機能の

コールによって生じる要求の識別子(要求id)が利用できるポイントで、イベントがトランスポート層にセットされる。このイベントはコマンドをサーバのdbxエンジンに送る。メッセージは要求id、クライアントのホスト名、およびクライアントのプロセス間アドレスを含んでおり、これが要求を一意に識別する。遠隔ホストのdbxエンジンに送られたメッセージはサーバのメッセージ通過層にセットされるブレークポイント・セットをもたらす。このブレークポイントは要求id、クライアントのホスト名、およびクライアントプロセス間アドレスの一致についてチェックし、一致が見つかった場合には、一致したスレッドがクライアントのトリガ機能によって開始された要求を処理するスレッドとなる。次いで、ブレークポイントが関数への進入時にそのスレッドにセットされ、サーバは継続される。

【0064】クライアントのトリガ機能が「foo」である場合、ブレークポイントがセットされるサーバの機能も「foo」となる。しかしながら、クラス情報がサーバで利用できないので、「stop in member foo」コマンドが特定のスレッドのチェックで使用される。場合によっては、自動的に生成されるIDLコードに機能「foo」があり、生成コードを実行すると、ブレークポイントが決まる。ブレークポイントが決まると、IDL生成コード内であるかどうかを判定するためのチェックが行われる。生成コード内にいる場合、他のブレークポイントがセットされ、実行が継続される。

【0065】クライアントのトリガ機能が「foo」である場合、サーバのブレークポイントが「foo」になったとき、復帰トリガ・スタック・ポインタがセーブされる。復帰トリガは「dstep」が「foo」の終わりで実行されたときに実行を継続するために使用される。ユーザは「cont」コマンドを使用して、サーバから明示的に継続することもできる。いずれの場合でも、セーブされた復帰トリガ・スタック・ポインタを廃棄し、これがサーバに対する他のコールで使用されないようにする。復帰時に実行されるように生成されたポイントで、イベントがトランスポート層にセットされ、セーブされた復帰トリガはそのイベントで廃棄される。イベントがスレッドでフィルタされるので、適正な要求の復帰だけが復帰トリガを廃棄する。

【0066】図14を参照して上述したように、メッセージをローカル・ホスト520のdbxエンジン504から、遠隔ホスト522のdbxエンジン512へ送って、サーバ516にブレークポイントをセットする。ブレークポイント・メッセージが送られた後、クライアント508が継続されるので、遠隔呼出しがクライアント508からサーバ516へ進む。しかしながら、ブレークポイント・メッセージを遠隔ホスト522のdbxエンジン512が受け取って、処理してから、遠隔呼出し

がサーバ516に到達するという保証はない。ブレークポイントが実際にセットされてから、遠隔呼出しが行われることを保証するために、ローカル・ホスト520のdbxエンジン504は停止し、遠隔ホスト522のdbxエンジン512からのメッセージを待つ。遠隔ホスト522のdbxエンジン512はサーバ516にブレークポイントをセットすると、メッセージをローカル・ホスト520のdbxエンジン504に送って、クライアント508を継続する。

【0067】上記で明らかにした付加的な変更／拡張を説明する。

【0068】あるdbxエンジンが他のdbxエンジンと通信することを可能とする変更は、メッセージを他のdbxエンジン（ターゲットdbxエンジンと呼ぶ）へ送るために、特定のToolTalkメッセージ（「rcmd」メッセージと呼ぶ）をdbxエンジンからデバッガGUIに渡すことができるようにするデバッガGUIおよびdbxエンジンの両方に対する変更を含んでいる。デバッガGUIはユーザからのコマンドを受け入れ、これを特定のdbxエンジンへ送る。「rcmd」メッセージはターゲットdbxエンジンが作動しているホストマシンの名前、ターゲットdbxエンジンがデバッグしているプロセスのプロセス識別子（pid）、およびターゲットdbxエンジンに対するメッセージを含んでいる。デバッガGUIはdbxエンジンのリストを維持しており、このリストはdbxエンジンが作動しているホストの名前、およびデバッグされているプロセスのpidを含んでいる。デバッガGUIは「rcmd」メッセージを取得すると、指定されたホストマシンで作動しており、所与のpidを有するプロセスをデバッグしているdbxエンジンに対するdbxエンジンのリストに到達する。ターゲットdbxエンジンに対するメッセージは次いで、そのdbxエンジンに送り出される。

【0069】デバッガGUIが特定のdbxエンジンにフォーカスするのを可能とする変更は、デバッガGUIに送信dbxエンジンに対するアテンションのフォーカスを変更させるために、特定のToolTalkメッセージ（「attention」メッセージと呼ぶ）をdbxエンジンからデバッガGUIに渡すことができるようにするデバッガGUIおよびdbxエンジンの両方に対する変更を含んでいる。デバッガGUIは特定のdbxエンジンに関連する表示を維持している。たとえば、特定のdbxエンジンがデバッグしているプロセスに対するソース・プログラムがデバッガGUIによって表示される。デバッガGUIは1組の表示を有しており、一度に1つのdbxエンジンからの情報を示すことができる。「attention」メッセージはデバッガGUIがその表示を現行dbxエンジンに対する情報から「attention」メッセージを送っているdbxエンジンの情報に変更することを示す。

【0070】デバッガGUIからすべての活動dbxエンジンのリストを取得できるようにするのに必要な変更は、デバッガGUIに各dbxエンジンが作動しているホスト名、およびそのdbxエンジンがデバッグしているプロセスのプロセス識別子（pid）のリストを返させるために、特定のToolTalkメッセージ（「get dbx engines」メッセージと呼ぶ）をdbxエンジンからデバッガGUIに渡すことができるようにするデバッガGUIおよびdbxエンジンの両方に対する変更を含んでいる。デバッガGUIはdbxエンジンが作動しているホストの名前、およびそのdbxエンジンがデバッグしているプロセスのpidを含んでいるすべてのdbxエンジンのリストを維持している。デバッガGUIは「get dbx engines」メッセージを受け取ると、ホストの名前およびデバッグされているプロセスのpidを各dbxエンジンに関して実行し、それをget dbx enginesメッセージを送ったdbxエンジンに返送する。

【0071】IV. 分散デバッガ使用方法

本発明のdoeDebuggerを作成するのに必要なSPARCworksデバッガ・システムに対する変更／拡張を説明してきたが、doeDebuggerを使用する方法をここで説明する。

【0072】図10を参照して、doeDebuggerの作動220を説明する。まず、ローカルマシンのプログラマー／開発者はdoeDebuggerを起動する（222）。ターゲット・プログラムが示され（224）、「dstep」コマンドが希望する機能に対して指定される（226）。doeDebuggerは標準「step」コマンドを実行し（228）、doeDebuggerはターゲット・インプリメンテーションがローカルか遠隔かを決定しようと試みる（230）。ターゲットが遠隔であると認識すると（IDLが生成した「remotesurrogate code」を認識することによって）、「find server」機能を実行して（232）、ターゲット・インプリメンテーションのホストidおよびpidを見つけ出す。ローカルdbxエンジンはコマンドを発行して、見つかったホストにdbxエンジンを作成し（234）、ブロックし、応答を待つ。見つかったホストはターゲット・インプリメンテーションに接続されているdbxエンジンがあるかどうかを判定する（236）。すでにdbxエンジンが作動している場合には（250）、見つかったホストのサーバは復帰メッセージを呼出し元クライアント側dbxエンジンに送る（252）。クライアント側dbxエンジンはメッセージを受け取って、ブロックを解除する（254）。クライアント側エンジンは次いで、メッセージをサーバのdbxエンジンに送って、指定された機能に一時ブレークポイントをセットする（262）（図11において）。図11を続けると、サーバd

dbxエンジンはコマンドを実行して、ターゲット機能にブレークポイントをセットし(264)、復帰トリガ・スタック・ポインタをセーブする(266)。サーバdbxエンジンはターゲット・インプリメンテーションを「継続」する(268)。その後、ターゲット・インプリメンテーションは指定されたブレークポイントにヒットし(270)、サーバdbxエンジンはブレークポイント処理し、クライアント・ホストのデバッグGUIにメッセージを送って、サーバdbxエンジンにフォーカスする(272)。デバッグGUIを使用しているプログラマはこれで、遠隔機能をこれがあたかもクライアント・ホストにあるような状態でデバッグできるようになる(274)。その後、システムはデバッグ・セッションが終わったかどうか(すなわち、quitコマンドを受け取ったかどうか)を調べるためにチェックを行い(276)、終わっていない場合には(278)、デバッグ・セッションを継続する(282)。quitコマンドを受け取っている場合には(280)、遠隔dbxエンジンは終了し、ターゲット・プロセスを接続解除し(284)、セッションを終わらせる(286)。

【0073】図10のブロック236に戻って、見つかったサーバ・ホストで作動しているdbxエンジンがない場合には(238)、サーバはクライアント・ホストのdbxエンジンに、dbxエンジンが作動していないことを伝える(240)。ヘルパ・オブジェクトを使用しているクライアントdbxエンジンはクライアント側dbxWrapperFactoryに対する要求を処理し、見つかったホストのサーバにdbxエンジンを作成する(242)。クライアント側dbxWrapperFactoryオブジェクトはサーバ側dbxWrapperFactoryインプリメンテーションをコールし(244)、dbxエンジンが見つかったホストのサーバに作成され、ターゲット機能のインプリメンテーションに接続する(246)。作成され、ターゲット機能に接続したばかりのこのdbxエンジンが起動され(248)、復帰メッセージがクライアント側dbxエンジンに送られ(252)、ブロック254および図11のブロックに関して上述したように、プロセスがこのポイントから継続する。

【0074】図10のブロック246に示した「遠隔dbxエンジン作成および接続」プロセスを図12を参照して詳細に説明する。図12において、「作成」プロセスが開始され(302)、クライアント側dbxエンジンがローカル・ヘルパ・プロセスをコールする(304)。ローカル・ヘルパ・プロセスは「create」コマンドをdbxWrapperFactoryオブジェクトに対して発行する(306)。dbxWrapperFactoryオブジェクトはメッセージを見つかったホスト308のdbxWrapperFactoryインプリメンテーションに送り、このインプリメンテ

ーションは「create」コマンドを実行する(310)。dbxWrapperFactoryインプリメンテーションはdbxエンジンに対して「fork」および「exec」を行い(316)、新たに作成されたdbxエンジンからのメッセージを待つ(318)。新しいdbxエンジンが何らかの理由で完全に起動されていない場合(324)、「Failed_to_start」メッセージが返され(326)、dbxWrapperFactoryインプリメンテーションは終了する(320)。新しいdbxエンジンが完全に起動されている場合には(322)、新しいdbxエンジンはターゲット機能に接続するよう指示される(328)。新しいdbxエンジンが接続できない場合には(334)、メッセージ「Failed_to_attach」が返され(336)、dbxWrapperFactoryインプリメンテーションは終了する(320)。新しいdbxエンジンが接続できる場合には(332)、メッセージ「Attached_and_running」が返され、dbxWrapperFactoryインプリメンテーションは終了する(320)。ターゲット・サーバへの「接続の失敗」が要件に接続するそれ自体の許可を有しているサーバによって生じることがあり、この場合、接続のための他の機構が必要となることがあることに留意されたい。

【0075】図11のブロック284に示した「遠隔dbxエンジン終了および接続解除」プロセスを、図13を参照して詳細に説明する。図13において、プログラマ/開発者がデバッグGUIに対して「Quit」コマンドを発行し(402)、デバッグGUIがこのコマンドをローカルdbxエンジンに渡すことによって「終了」プロセスが開始される(404)。ローカルdbxエンジンは「QuitSession」メッセージをヘルパ・プロセスを介してdbxWrapperFactoryに送り、デバッグ・セッションを終了する(406)。「QuitSession」コマンドにより、関与しているホストの各々のdbxWrapperFactoryオブジェクトは各dbxエンジンに信号を送る(408)、このdbxエンジンはデバッグ・セッションの一部である(410)。各dbxエンジンは送られた信号に対する信号ハンドラを有しており、このハンドラは信号がdbxWrapperFactoryから送られているのかどうかを調べ、そうである場合には、dbxエンジンはデバッグしているプロセスとの接続を解除し(412)、終了する。他の実施の形態はdbxエンジンをそのターゲット機能から接続解除するステップ(414)や、「dbx_debugger_detached_and_deleted」などのメッセージをクライアント側dbxエンジンに返すステップ(416)などの付加的なステップを含んでいることができる。

【0076】dbxWrapperを実施するインタフェースを図15に示す。当分野の技術には、各種のインプリメンテーションがこの機能に対して行えることが認識されよう。

【0077】本発明を特定のオペレーティング・システム、プログラム・コード、ならびにオブジェクトおよびオブジェクト・リファレンスの定義に関して説明してきたが、本発明を所与のオペレーティング環境内の、あるいは異なるオペレーティング・システムまたはオブジェクト・システム環境の各種の変形のうちの任意のもので実施できることが当分野の技術者には理解されよう。同様に、図示の特定のクライアントおよびサーバの構成または組合せは、本発明を使用することのできるクライアントとサーバの多くのこのような構成、ならびにオブジェクトとサブオブジェクトの関係の代表的なものに過ぎない。さらに、図面が説明のためのものに過ぎず、本発明の限定事項と取るべきではないことが理解されよう。遠隔dbxエンジンの他の機能を備えたクライアント側デバッガGUIとの付加的な組合せは、ターゲット・オブジェクトにフレンドリ・ユーザ・インタフェースをもたらしグラフィカル・ユーザ・インタフェース（GUI）エージェントとのdbxエンジンの組合せ、ユーザの既知の選択に基づいて遠隔要求を変更する人工知能エージェントとの遠隔dbxエンジンの組合せ、遠隔要求に対する回答をキャッシュするキャッシュ・プログラムとの遠隔dbxエンジンの組合せ、数人のユーザからの入力をマージして、これをターゲットに送るテレコンファレンス・アプリケーションとの遠隔dbxエンジンの組合せ、あるいはマルチメディア・システムの多数のオーディオおよびビデオ・アクセス・エージェントとの遠隔dbxエンジンの組合せを含んでいる。これらの考えられるdbxエンジンとデバッガGUIの組合せは本明細書で開示した遠隔デバッガ機能の可能な用途を限定することを意図したものではなく、当分野の技術者が例と認識する例を表しているに過ぎない。doeDebuggerの発明の範囲は、したがって、首記の特許請求の範囲、ならびに特許請求の範囲の対象となる同等物の全範囲を参照することによって決定されるものである。

【図面の簡単な説明】

【図1】汎用コンピュータおよび関連装置を示す図である。

【図2】分散コンピュータ・システムを示す図である。

【図3】ユーザ、クライアント・アプリケーション、オブジェクト・リファレンス、オブジェクト・インプリメンテーション、およびオブジェクト・リファレンス作成プログラムの関係を示す、複数のマシンを備えたクライアント・サーバ・システム構成の図である。

【図4】単一のマシンを使用したクライアント・サーバ構成の図である。

【図5】SPARCworksデバッガを示す図であ

る。

【図6】クライアント・アプリケーション、オブジェクト・リファレンスおよびSPARCworksデバッガの間の関係を示す図である。

【図7】複数のサーバにアクセスする必要のある分散オブジェクト環境（DOE）クライアント・アプリケーションの例を示す図である。

【図8】ユーザ・コードとDOEクライアントまたはサーバの基礎DOE機構の間の分割を示す図である。

【図9】doeDebugger構成を示す図である。

【図10】分散環境のdoeDebuggerの作動を示す流れ図である。

【図11】dbxエンジン間プロセスを示す流れ図である。

【図12】doeDebugger作成および接続プロセス（図10の流れ図のブロック246）の作動を示す流れ図である。

【図13】doeDebugger終了および接続解除プロセス（図11の流れ図のブロック284）の作動を示す流れ図である。

【図14】doeDebugger、dbxエンジン、ラップ・サービスおよびクライアント・アプリケーションおよびサーバ・オブジェクト・インプリメンテーションの間の関係を示す図である。

【図15】dbxWrapperの典型的なインタフェースを示す図である。

【符号の説明】

- | | |
|-----------------------|---------------|
| 1 コンピュータ、 | 2 入出力（「I/O」）部 |
| 3 中央処理装置（「CPU」）、 | 4 メモリ部 |
| 5 キーボード、 | 6 表示装置 |
| 7 CD-ROM駆動装置、 | 8 CD-ROM媒体 |
| 9 ディスク記憶装置、 | 10 プログラム |
| 20、22、24 コンピュータ | |
| 26 通信リンク | |
| 28 共用メモリ装置 | |
| 30 ユーザ | |
| 34 クライアント・アプリケーション | |
| 36 オブジェクト・リファレンス | |
| 46 オブジェクト・インプリメンテーション | |
| 50 サーバ | |
| 92 ホストマシン | |
| 96 ToolTalk通信リンク | |
| 98 dbxエンジン | |
| 112 赤ホスト | |
| 130 白ホスト | |
| 142 ローカル・ホスト | |
| 144 機能「foo」 | |
| 148 生成コード | |

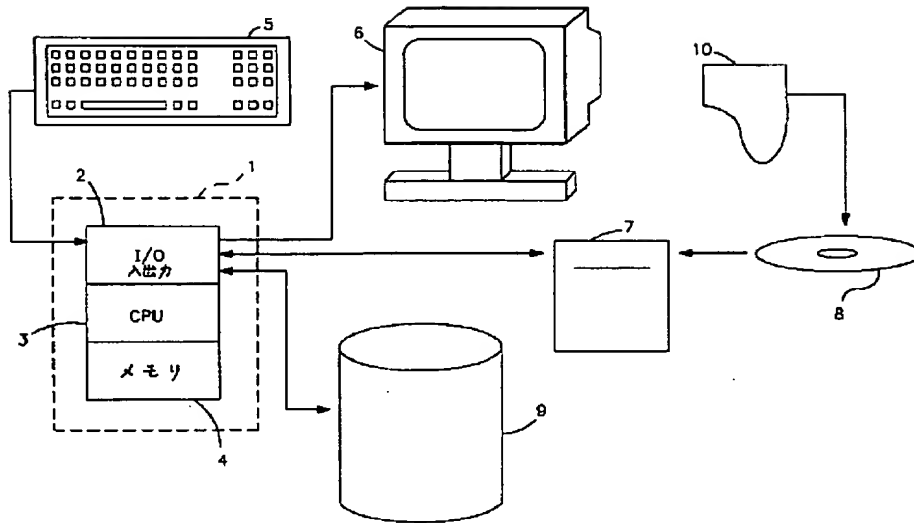
31

32

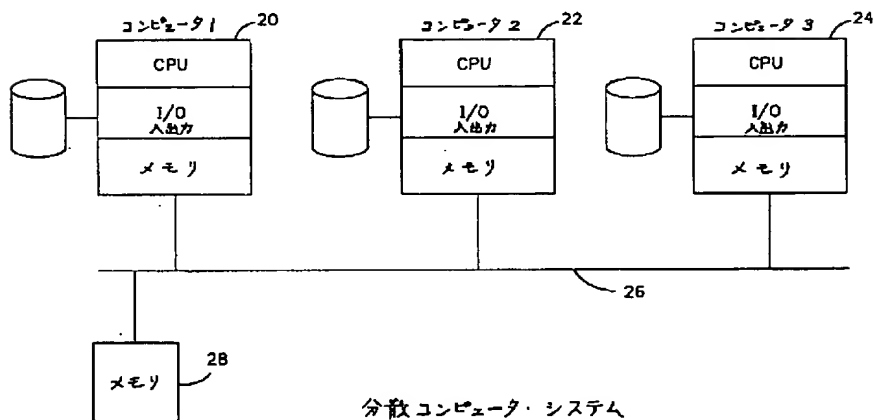
502 デバッガGUI
506 ヘルパ・プロセス

510 ラッパ・サーバ
522 遠隔ホスト

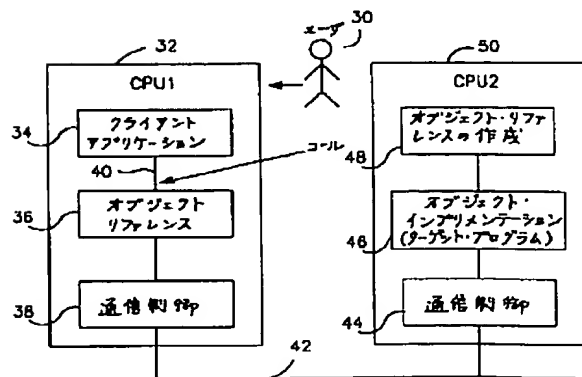
【図1】



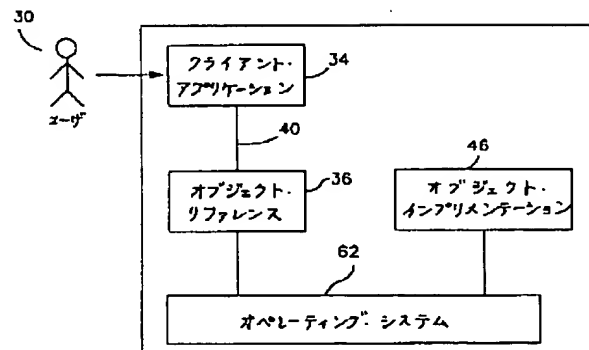
【図2】



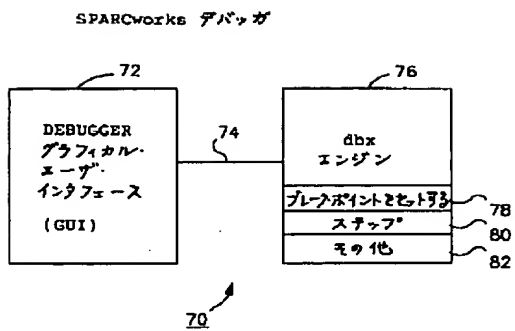
【図3】



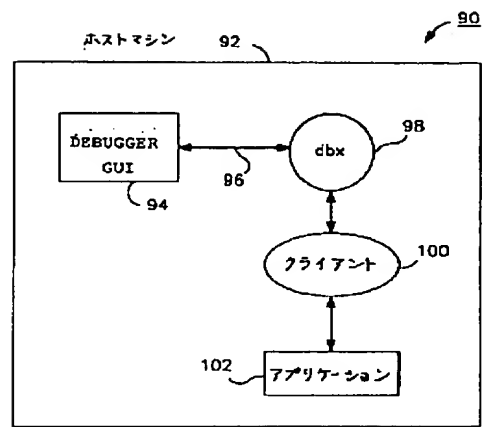
【図4】



【図 5】

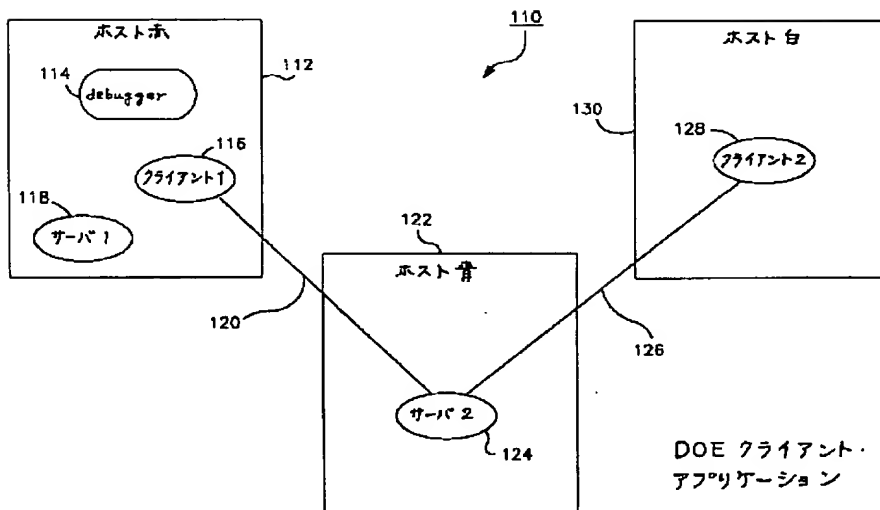


【図 6】

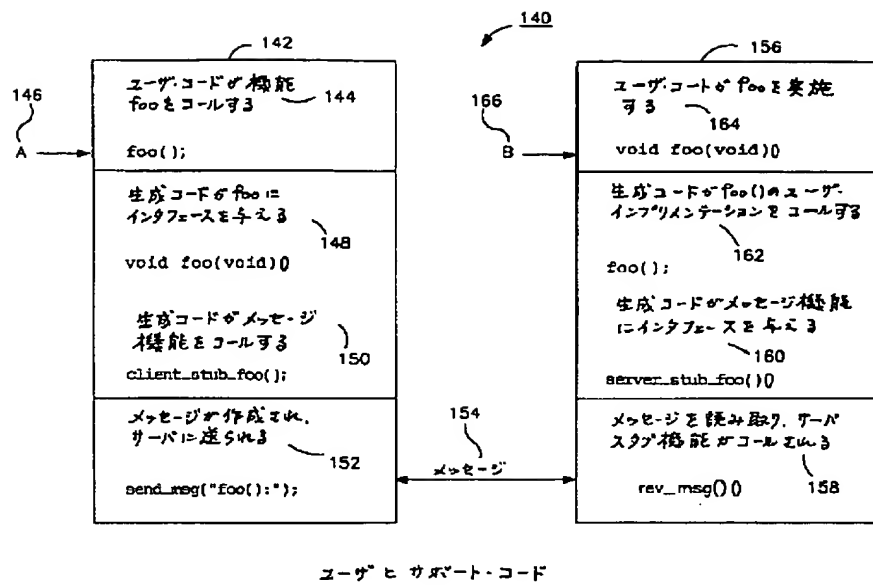


基本的な Debugger の作動

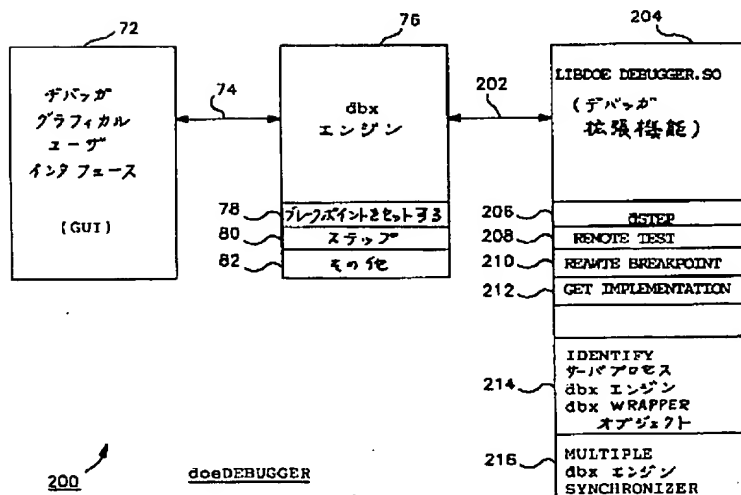
【図 7】



【図 8】

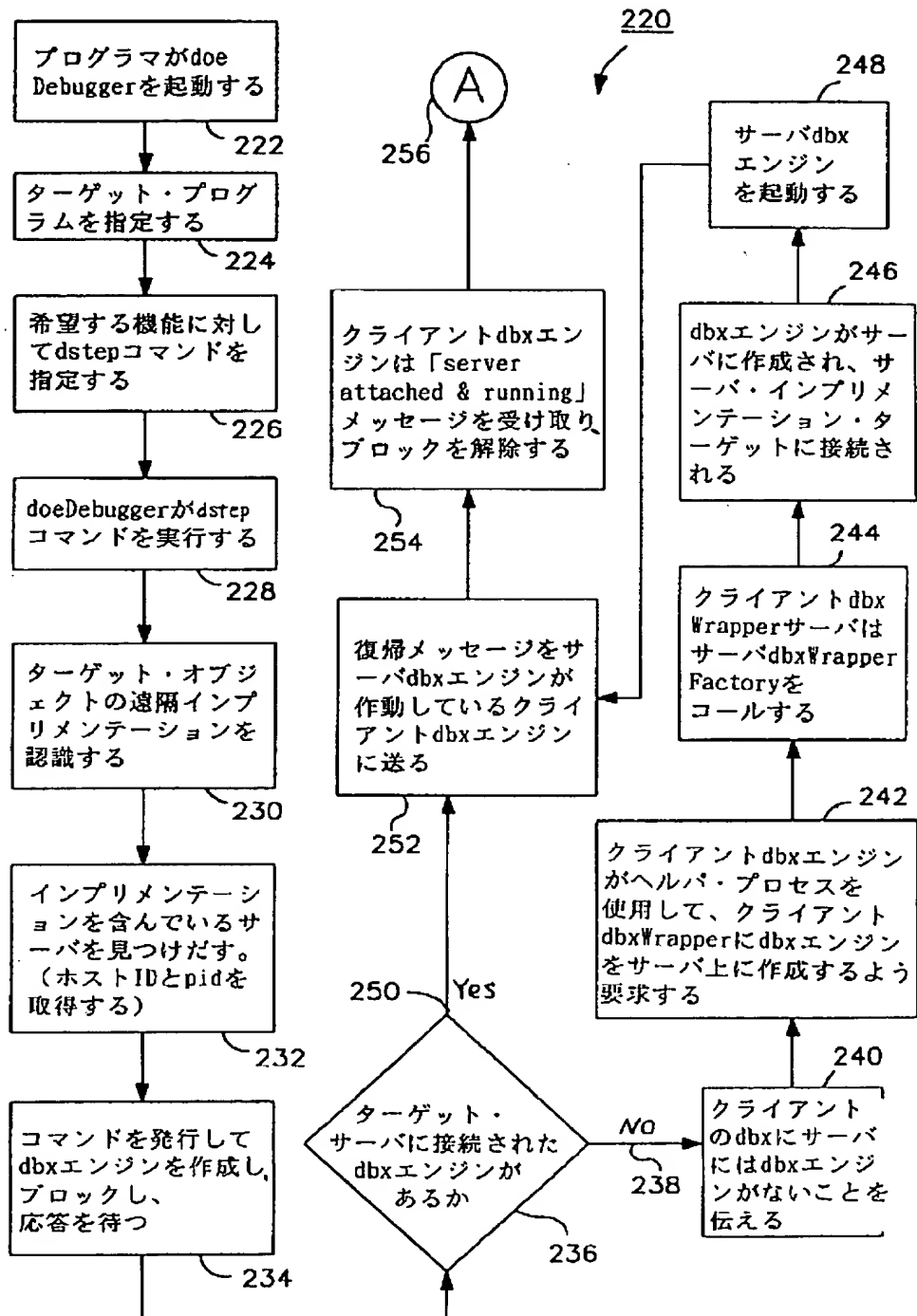


【図 9】

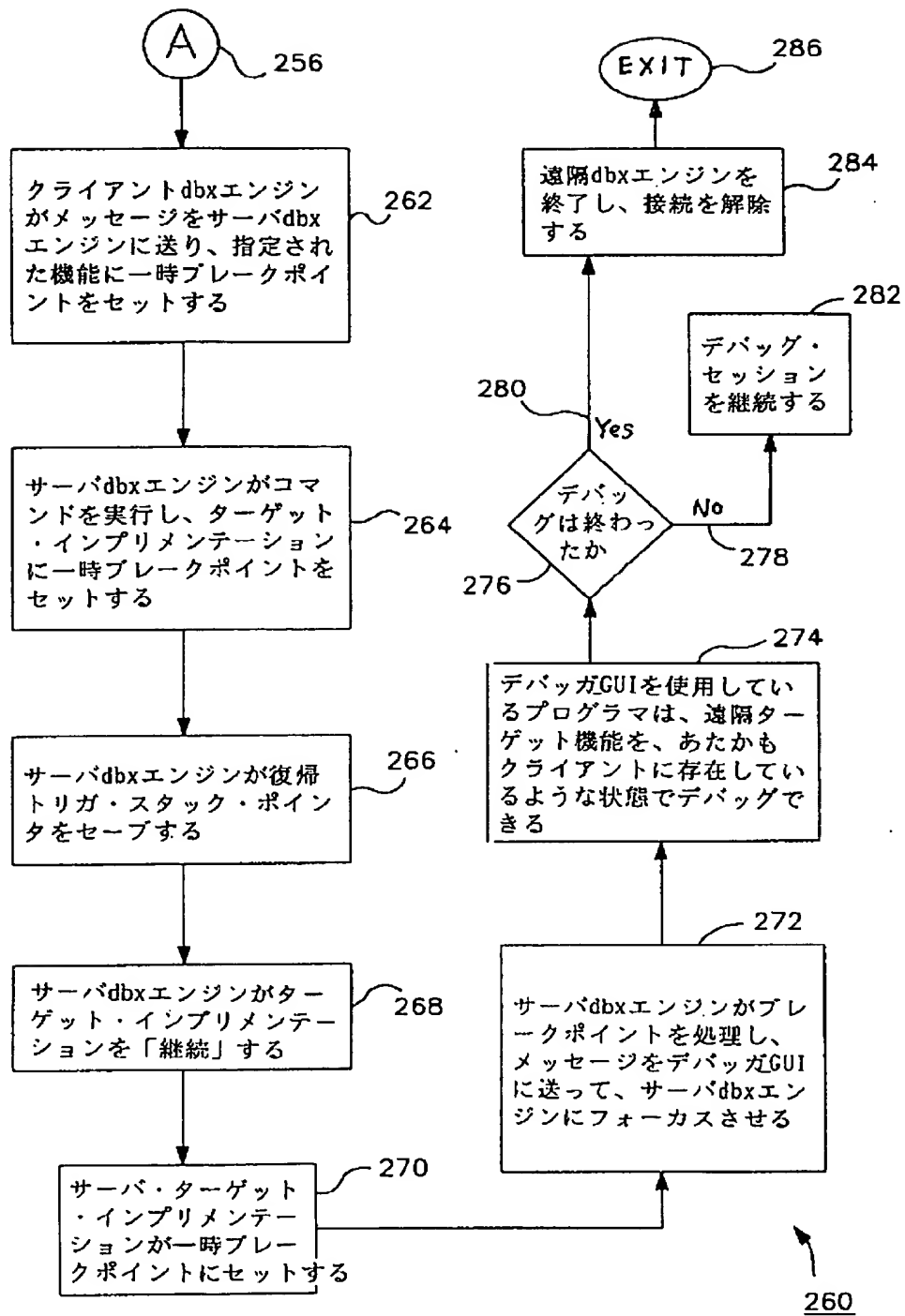


【図10】

doe Debugger の作動

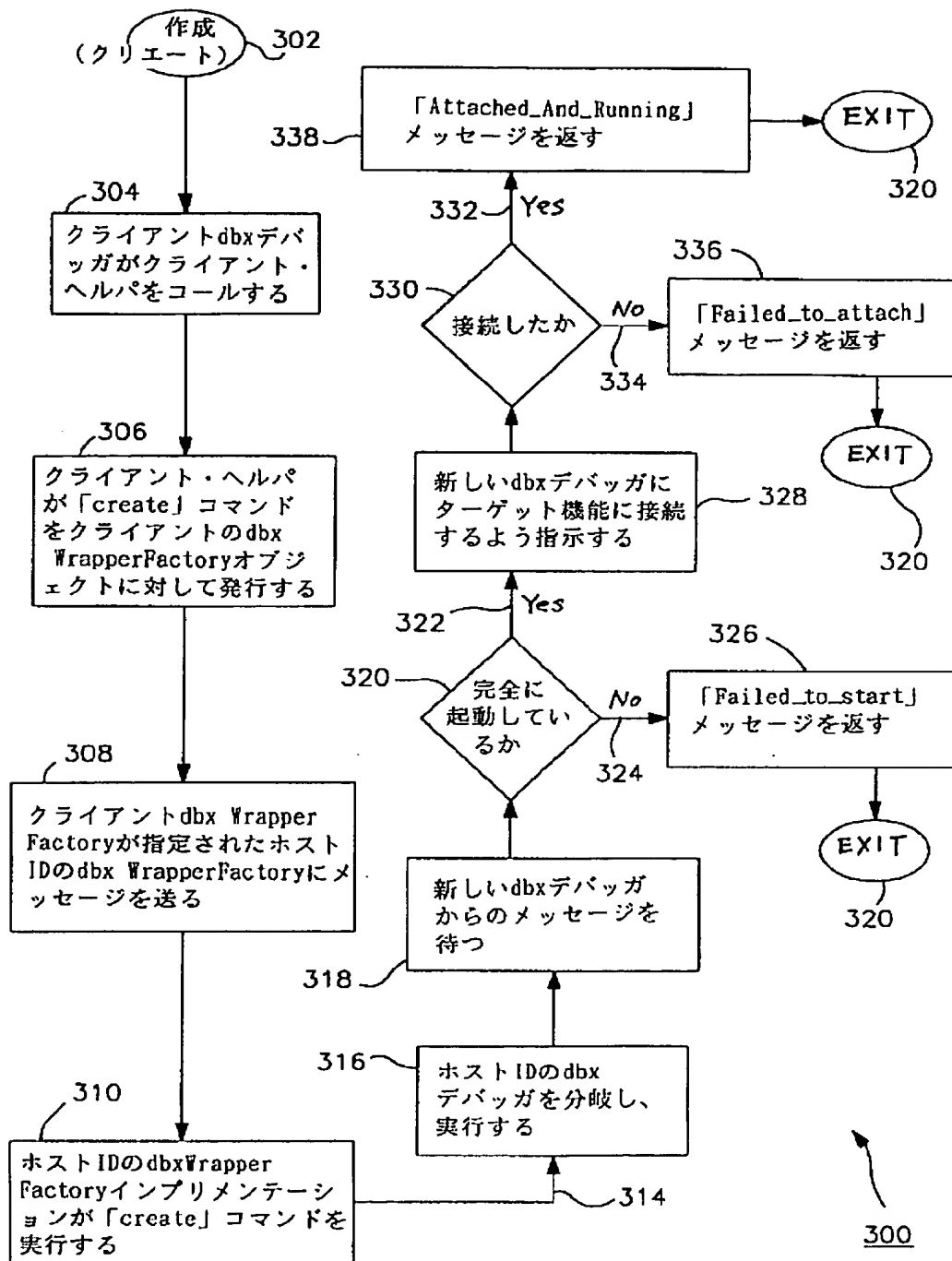


【図 11】



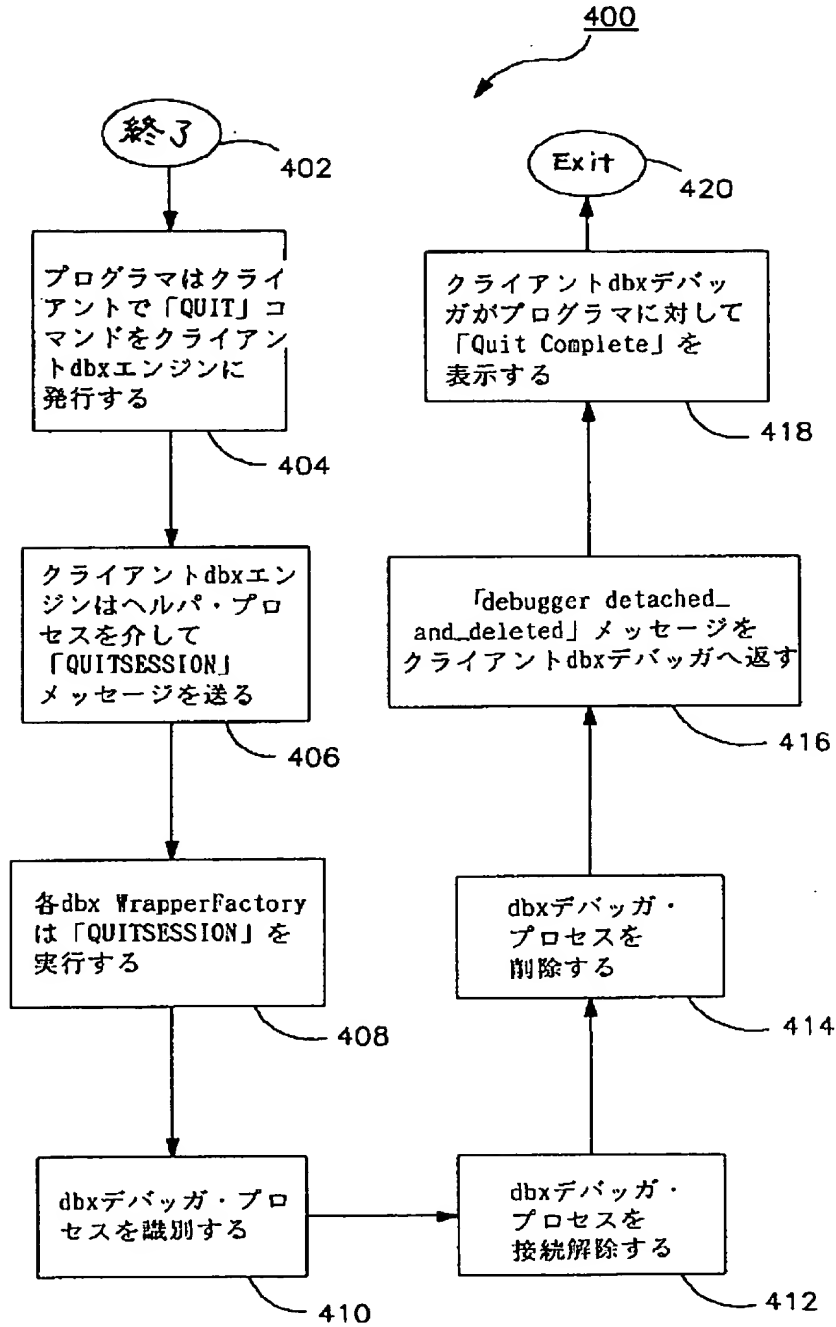
【図12】

遠隔dbxエンジンの作成と接続

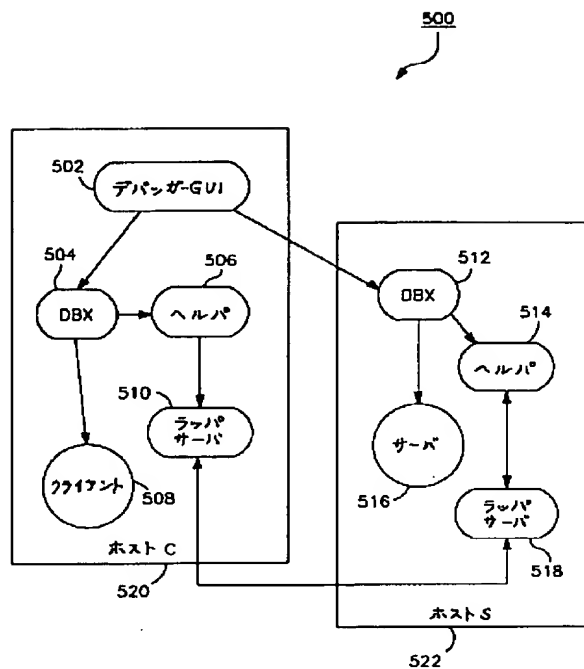


【図 1 3】

遠隔dbxエンジンの終了と接続解除



【図 14】



【図 15】

DbxWrapper インタフェース

```

enum DbxWrapperFailedReasons {
    FAILED_TO_START,
    FAILED_TO_ATTACH,
    ALREADY_BEING_DEBUGGED,
    PERMISSION_DENIED
};

exception DbxWrapperFailed {
    DbxWrapperFailedReason reason;
};

interface DbxWrapper {
    void SynchronizeDbxEngine();
    void Quit();
};

interface DbxWrapperFactory {
    DbxWrapper Create(
        in object servant, // デバッグするオブジェクト
        in string path,    // dbx プログラムパス
        in string dbxEngine, // dbx プログラム名
        in string display,  // Xディスプレイ接続する
        in string arguments // tool Talk アドレスを指定
    )raises(DbxWrapperFailed);

    void DbxEngineAttached();
};

```

フロントページの続き

(72)発明者 ジョン・エイ・マサミツ
 アメリカ合衆国 94550 カリフォルニア
 州・リバモア・クリーク ロード・1873